1. Compare performance for single-cycle, multicycle, and pipelined using the following assumption.Calculate average instruction times for the three architectures.
   - ✓ Instruction mix:24% loads,15% stores,15% branches,6% jumps,40% R-type
   - ✓ 200ps for memory access,100ps for ALU operation,50ps for register file R/W
   - ✓ For pipelined execution,one-quarter of loads are immediately followed by an instruction that uses the results
   - ✓ For pipelined execution,branch delay on misprediction is 2 clock cycle,and one-third of the branches are mispredicted
   - ✓ For pipelined execution,jumps always pay 1 full clock cycle of delay

   Sol:

   Single-cycle:

   Clock cycle time:

   200+50+100+200+50=600(ps)

   Instruction fetch -> instruction decode -> execution ->memory->write back

   Memory access(200ps)+register file R/W(50ps)+ALU operation(100ps)+memory access(200ps)+register file R/W(50ps)

   CPI = 1 (一個時脈週期內執行一個指令)

   Multi-cycle:

   Clock cycle time:200(ps)

   多重時脈的時脈週期取決於最長步驟的執行時間

   CPI = 24%*5+15%*4+15%*3+6%*3+40%*4=4.03

instruction	Step name					CPI
R-type	IF	ID/RF	execution		WB	4
lw	IF	ID/RF	Addr. calculation	MA	WB	5
sw	IF	ID/RF	Addr. calculation	MA		4
beq	IF	ID/RF	Compare reg.			3
j	IF	ID	Compose targetaddr.			3
Function unit	Instr. Mem.	Reg. file	ALU	Data Mem.	Res. file	

   Pipelined:

   Clock cycle :200(ps)

管線速率受限於最慢管線階段的執行時間

CPI:

Load:1/4*2+3/4*1=5/4=1.25

Store&ALU:1

Branch :1/3*3+2/3*1=5/3

Jump: 2

=>24%*1.25+15%*1+15%*5/3+6%*2=1.22

沒有使用載入資料的相依性=>讀取要花費"一個時脈"週期的時間

有　使用載入資料的相依性=>讀取要花費"兩個時脈"週期的時間

Ans:

Single cycle:1*600=600(ps)

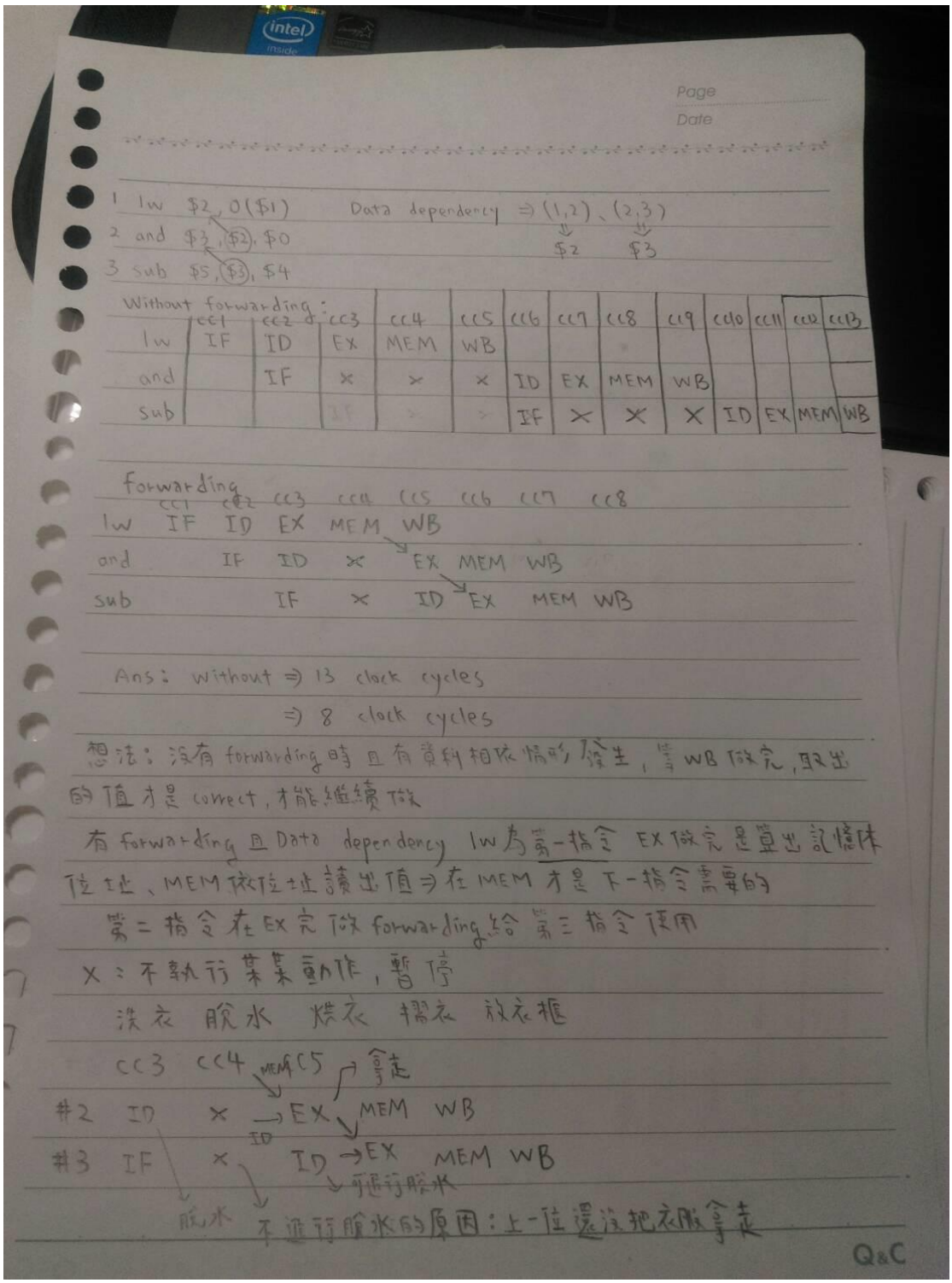
Multi-cycle:4.03*200=806(ps)

Pipelined:1.22*200=244(ps)

2. In a pipelined datapath,answer the following questions:
   (a) How to flush the instruction in the IF stages?(補:480)
   IF.Flush is added to zero the instruction field of the IF/ID pipeline register
   加入一個 IF.Flush 的控制線，將 IF/ID 管線指令暫存器的指令欄位設為 0
   (b) How to flush the instruction in the ID stages?
   把 ID/EX 設為 0,把當時所有 control line 清空
   (c) How to flush the instruction in the EX stages?
   把 EX/MEM 設為 0,把 EX/MEM 的控制訊號清空

3. how many clock cycles are required to execute the code sequence in the five stage pipeline without and with forwarding technique?

   lw $2,0($1)
   and $3,$2,$0
   sub $5,$3,$4



4. show the mips machine code for this loop(in decimal value).

MIPS code

```
Loop: lw   $t2, 16($s3)
      slt  $s0, $t3, $s1
      addi $sp, $sp, -20
      xor  $t4, $t0, $s1
      j    Exit
      beq  $s4, $t5, Loop
      srl  $s1, $a3, 5
Exit:
```

Name	Register number	Usage	Preserved on call?
$zero	0	the constant value 0	n.a.
$v0–$v1	2–3	values for results and expression evaluation	no
$a0–$a3	4–7	arguments	no
$t0–$t7	8–15	temporaries	no
$s0–$s7	16–23	saved	yes
$t8–$t9	24–25	more temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

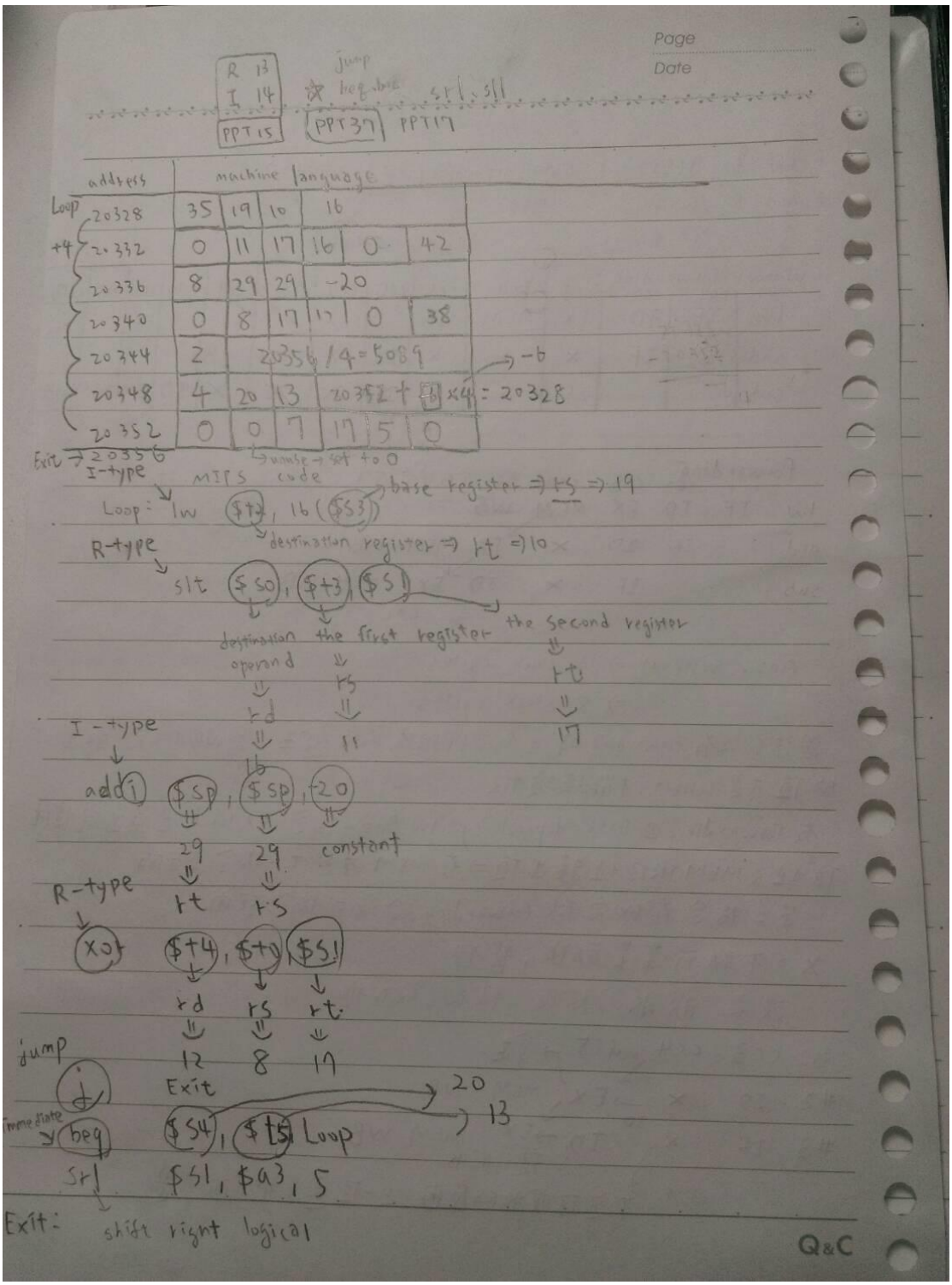
functional field:

xor: 38 srl: 0 slt: 42

opcode:

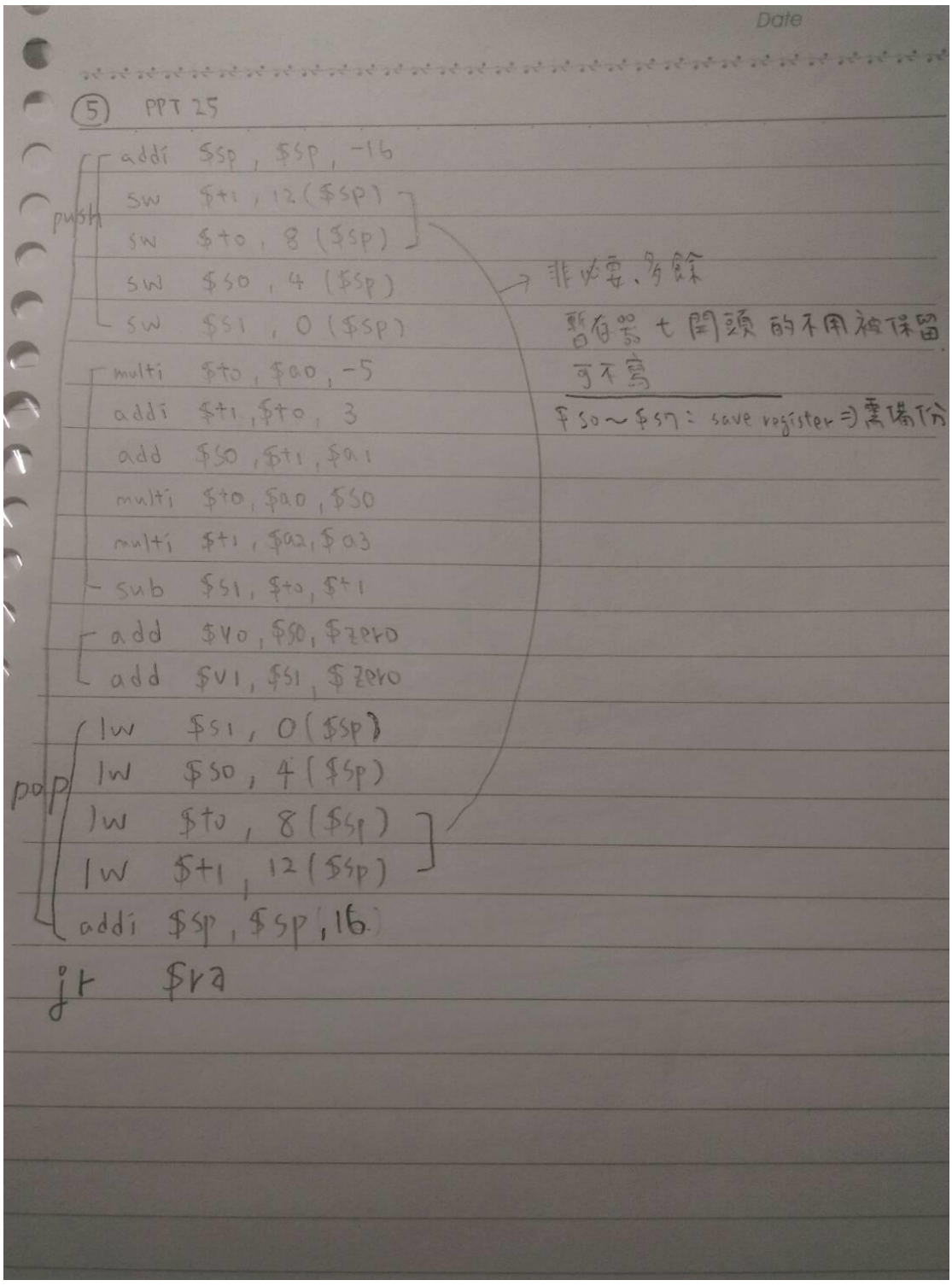
beq: 4 lw: 35

addi: 8 j: 2

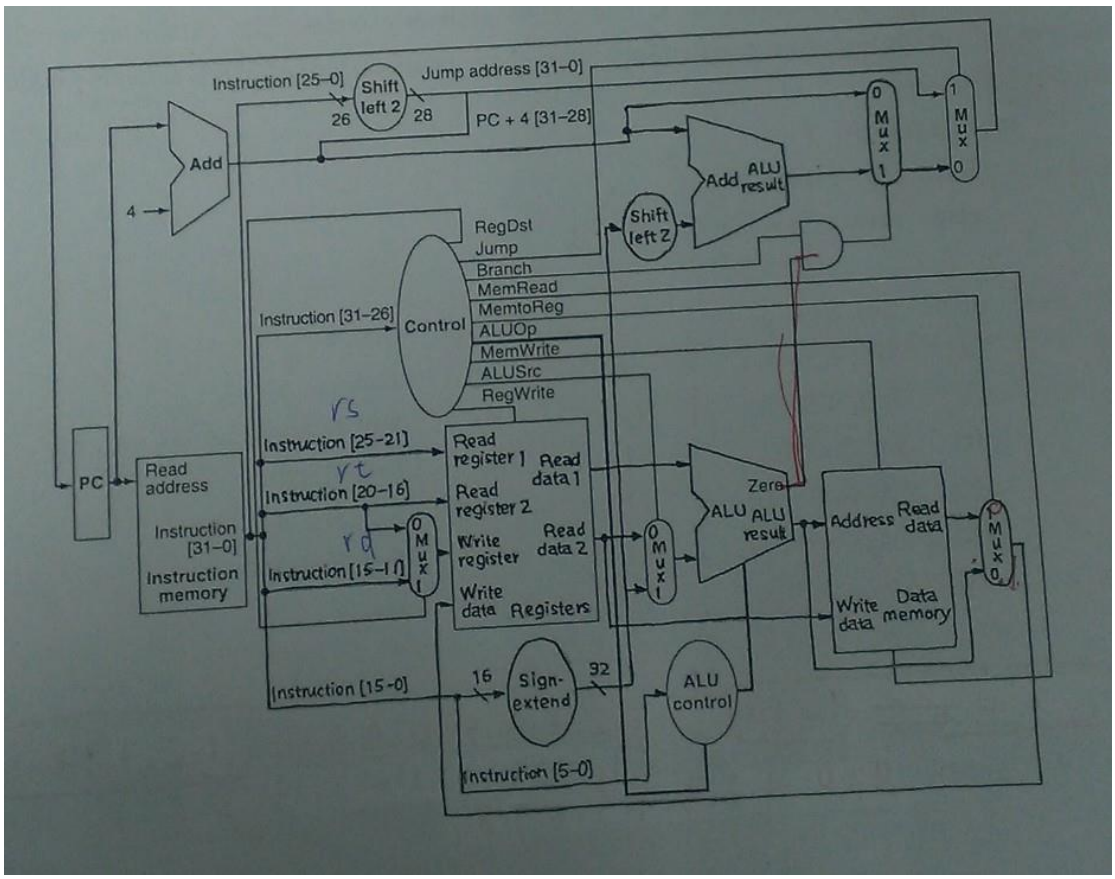


5. translate the following C code to MIPS assembly code.The parameter variables (a,b,c,d,m,n) corresponds to ($a0,$a1,$a2,$a3,$s0,$s1)

```
int example (int a, int b, int c, int d)
{
    int m, n;
    m = 3 - a × 5 + b;
    n = a × m - c × d;
    return m, n;
}
```


Date
⑤ PPT 25
addi $sp, $sp, -16
push
sw $t1, 12($sp)
sw $t0, 8($sp)
sw $s0, 4($sp)
sw $s1, 0($sp)
multi $t0, $a0, -5
addi $t1, $t0, 3
add $s0, $t1, $a1
multi $t0, $a0, $s0
multi $t1, $a2, $a3
sub $s1, $t0, $t1
add $v0, $s0, $zero
add $v1, $s1, $zero
pop
lw $s1, 0($sp)
lw $s0, 4($sp)
lw $t0, 8($sp)
lw $t1, 12($sp)
addi $sp, $sp, 16
jr $ra
非必要、多餘
暫存器七開頭的不用被保留
可不寫
$s0~$s7: save register 需備份

6.


Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
M u x
0
1
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
rs
Instruction [25–21]
rt
Instruction [20–16]
rd
Instruction [15–11]
PC
Read address
Instruction [31–0]
Instruction memory
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
Instruction [15–0]
16
Sign-extend
32
ALU control
Instruction [5–0]

Date 12/1

觀念

控制訊號	Effect when deasserted	1
RegDst	選擇要寫入的目的暫存器號碼來自指令(rt)欄位	來自(rd)欄位
RegWrite	×	允許資料寫入暫存器
ALUSrc	ALU的第二運算元的值取自暫存器檔案的第二輸出	取自指令較低16位元經過擴充後的值
PCSrc	choose (PC+4的值) 寫入PC	choose 分支目的位址 寫入PC
MemRead	×	允許要被讀取的資料輸出記憶體
MemWrite	×	允許要被寫入的資料輸入記憶體
MemtoReg	choose要寫入暫存器的值是來自ALU的計算結果	來自資料記憶體

✓ Branch = 0 ∵ R-type 指令並非分支指令

✓ ALUOp = 10 根據功能欄位

00 加法

01 減法

(a) ✓ ✓ ✓

Instruction	MemtoReg	ALUSrc	RegWrite	MemWrite	RegDst	MemRead	Jump	Branch
lw	1	1	1	0	0	1	0	0
sw	X	1	0	1	X	0	0	0
R-type	0	0	1	0	1	0	0	0
beq	X	0	0	0	X	0	0	1
j	X	X	0	0	X	0	1	0

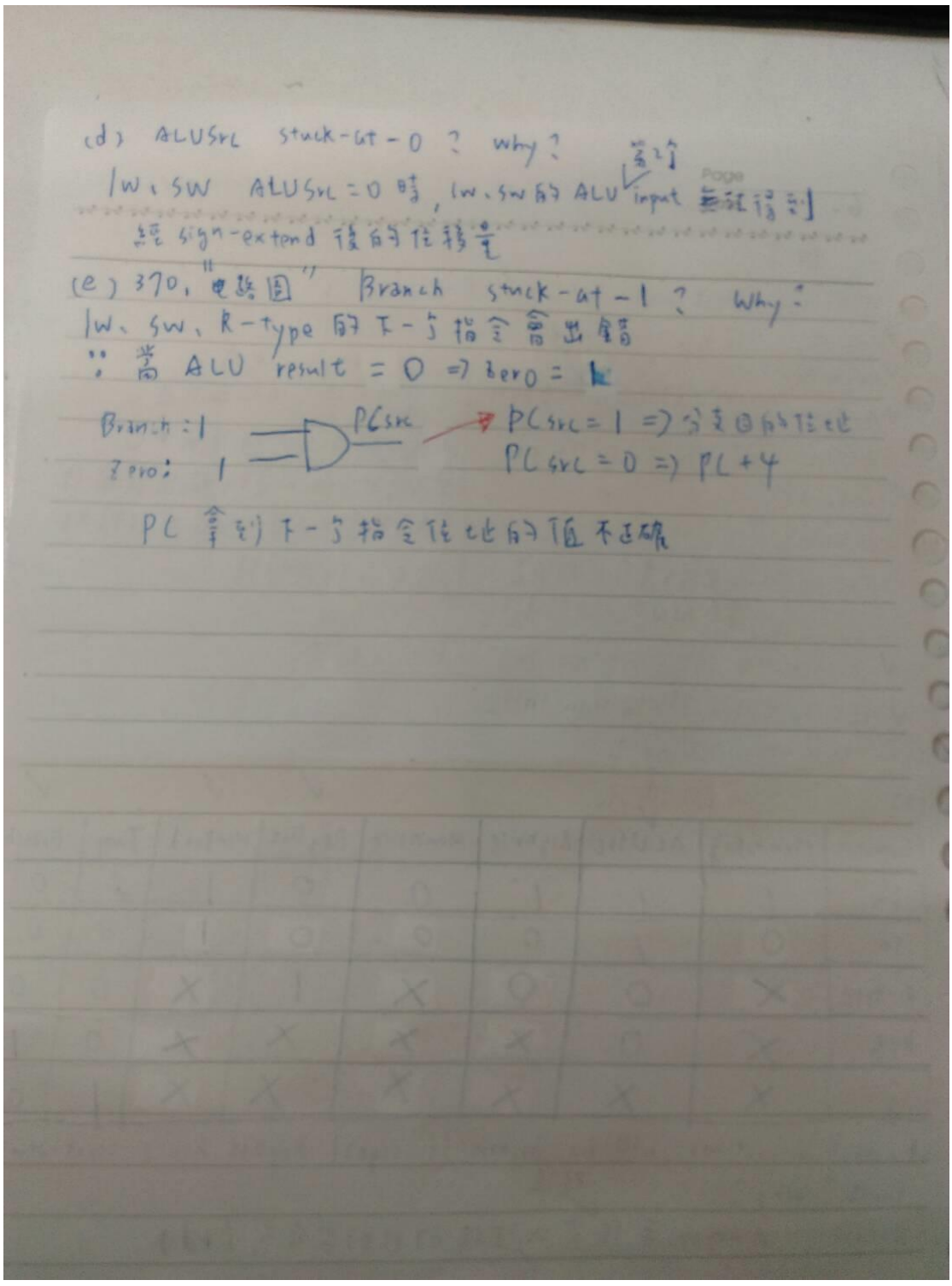
(b) Which instructions will be broken if signal RegDst has a stuck-at-0 fault? Why?

R-type, R-type 無法寫入正確的目的暫存器(rd)

(c) Which instructions will be broken if signal MemRead has a stuck-at-1 fault? Why?

sw, MemRead = 1 時, 會執行 lw, 所以 sw 還沒做完寫入記憶體的動作 ~~將資料~~ ~~無法進行寫入記憶體~~ Q&A

直從 Memory 讀出來的值 ~~未~~ 不正確



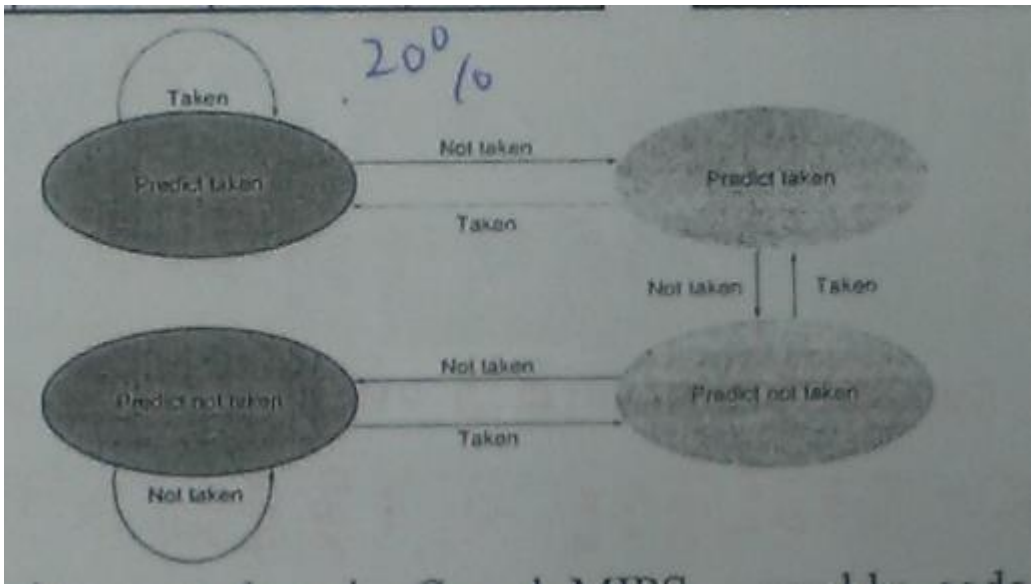
7.

(a)Give a sequence of branch outcomes that the accurancy of the 1-bit predictor is higher than the accuracy of the 2-bit predictor?

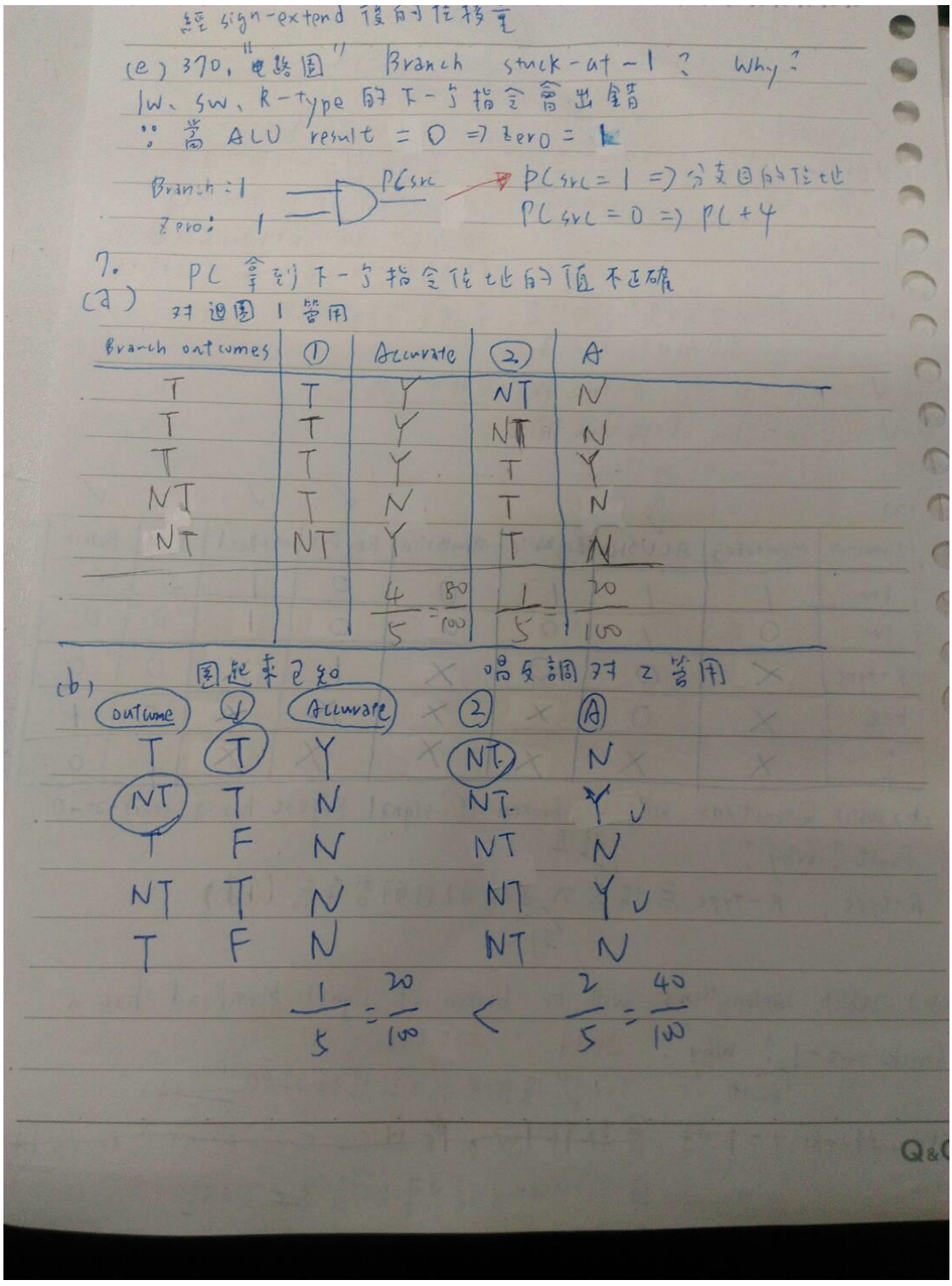
(b)Give a sequence of branch outcomes that the accurancy of the 2-bit predictor is

higher than the accuracy of the 1-bit predictor?

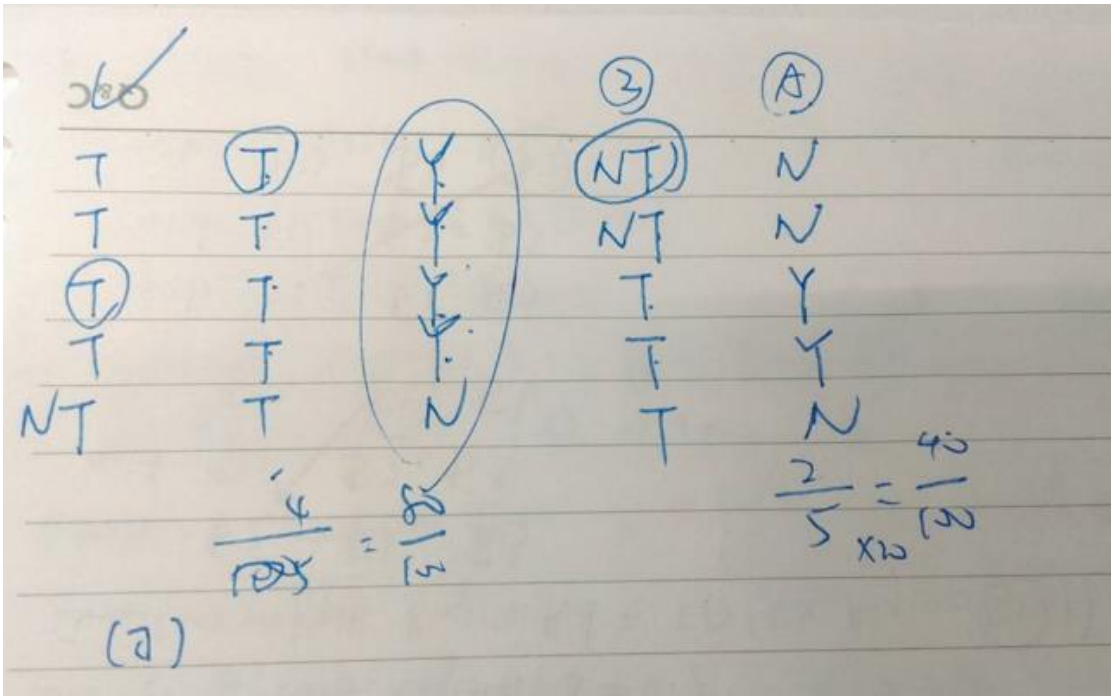
Ps: 1 bit 從 predict taken 開始 , 2 bit 從圖中左下開始



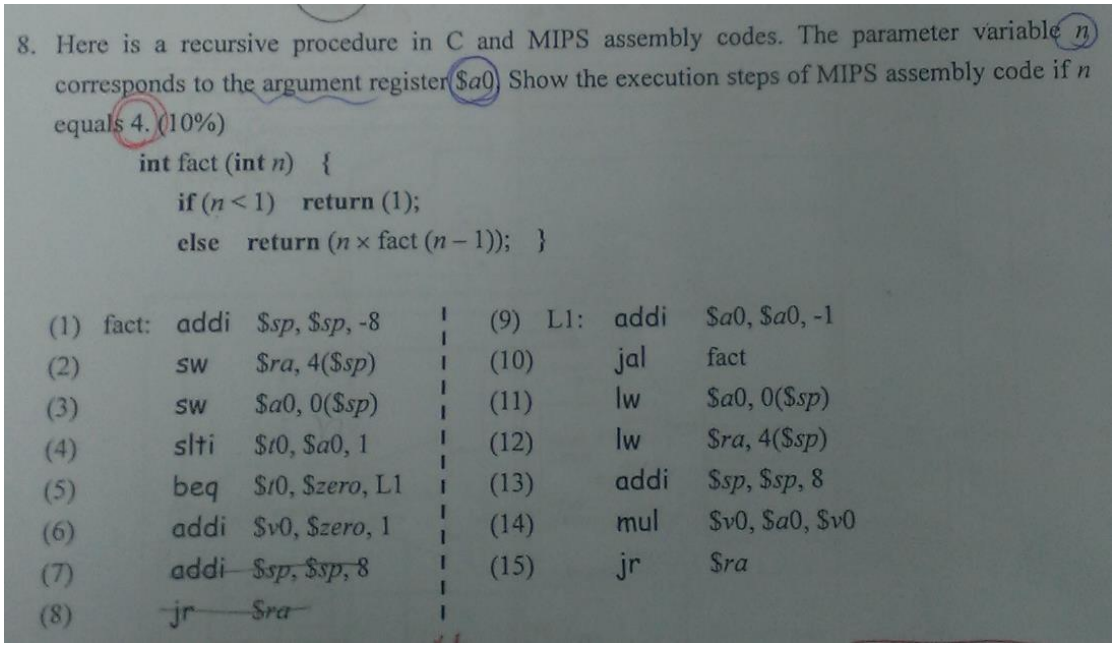
Ans:


經 sign-extend 後的位移量
(e) 370, "電路圖" Branch stuck-at-1 ? Why ?
lw、sw、R-type 的下一个指令會出錯
∵ 當 ALU result = 0 ⇒ zero = 1
Branch = 1
Zero = 1
PCSrc
PCSrc = 1 ⇒ 分支目的地址
PCSrc = 0 ⇒ PC + 4
7.
PC 拿到下一个指令位址的值不正確
(a)
對過圖 1 管用
Branch outcomes
①
Accurate
②
A
T T Y NT N
T T Y NT N
T T Y T Y
NT T N T N
NT NT Y T N
4/5 = 80/100
1/5 = 20/100
(b)
圖起來已知
唱反調對 2 管用
outcome
①
Accurate
②
A
T T Y NT N
NT T N NT Y
T F N NT N
NT T N NT Y
T F N NT N
1/5 = 20/100 < 2/5 = 40/100

另:



8.



8. Here is a recursive procedure in C and MIPS assembly codes. The parameter variable n corresponds to the argument register $a0. Show the execution steps of MIPS assembly code if n equals 4. (10%)

```
int fact (int n)  {
    if (n < 1)  return (1);
    else  return (n × fact (n – 1));  }
```

```
(1) fact: addi  $sp, $sp, -8        (9)  L1: addi  $a0, $a0, -1
(2)       sw    $ra, 4($sp)         (10)      jal   fact
(3)       sw    $a0, 0($sp)         (11)      lw    $a0, 0($sp)
(4)       slti  $t0, $a0, 1         (12)      lw    $ra, 4($sp)
(5)       beq   $t0, $zero, L1      (13)      addi  $sp, $sp, 8
(6)       addi  $v0, $zero, 1       (14)      mul   $v0, $a0, $v0
(7)       addi  $sp, $sp, 8         (15)      jr    $ra
(8)       jr    $ra
```

觀念:

(1)~(5): 給一個 space 放 register , save saved register,if 判斷是否 argument(a0) 小於 1 =>slti $t0,$a0,1 , beq $t0,$zero,L1

有則執行 return(1) (6):return 1(7)pop 2 items from stack(8):return to caller

沒有則執行 else (9),(10)

最後 restore 和 return to caller 的動作(11)(12)(13)(14)

stack pointer ($sp)

$sp: 指向堆疊區域"最新資料"所配置位址的暫存器

push: $sp is decreased
pop: $sp is increased

程序呼叫:
六步驟:
1. 將引數放在程序(callee)可以存取的地方
// set arguments($a0~$a3)

2. 將控制權轉移給程序(callee)
// jal B

*3. 取得程序(callee)所需的儲存資源
// save saved registers ($s0~$s7)

4. 執行所指定的工作
// execute specified jobs

*5. 將所得結果放在呼叫程式(caller)可以存取得到的地方
// set return values ($v0~$v1)
    restore saved registers ($s0~$s7)

6. 將控制權將交回給呼叫程序(caller)
// jr $ra

指令:

jal(jump and link)程序呼叫指令:跳到程序起始位址，同時將下一個指令的位址(PC+4)儲存在暫存器$ra
jal ProcedureAddress

jr : 程序返回
jr $ra

$sp 堆疊指標 用來儲存被呼叫者所需要的暫存器位址

由高位址(high address)向低位址(low address)成長 push=>減法 pop=>加法

ppt26

slti:slti $t0,$s2,10 # t0 = 1 if $s2 < 10 (ppt21)

0000 0000 0000 0000 0000 0000 0000 0001

1111 1111 1111 1111 1111 1111 1111 1110(1 的補數)

1111 1111 1111 1111 1111 1111 1111 1111(2 的補數)

Ans:

N = 4 => 要算 4!

(1,2,3,4,5,9,10) *4 → (1,2,3,4,5,6,7,8) →(11,12,13,14,15)*4

<補充> 支援 bne 指令的 datapath

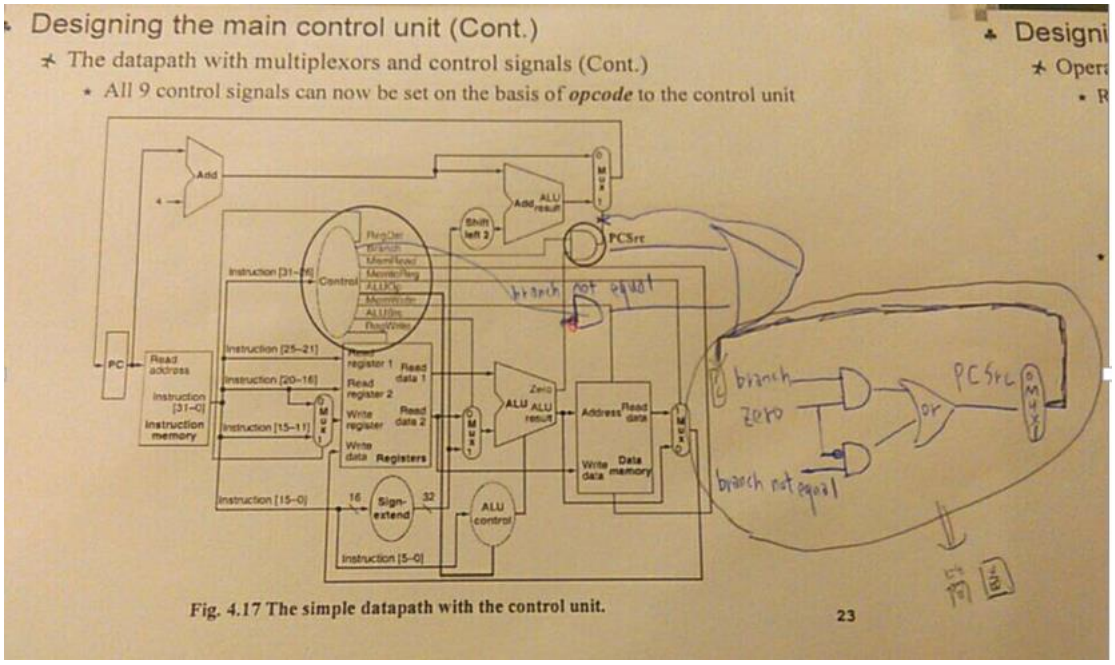


Fig. 4.17 The simple datapath with the control unit.

23